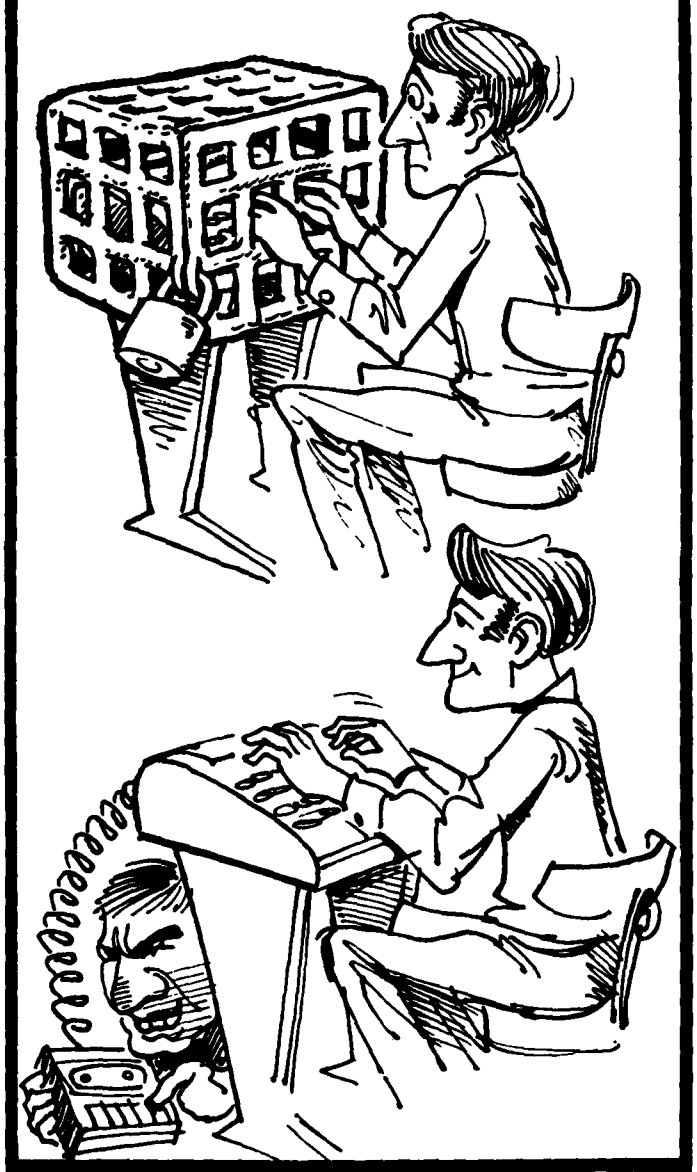
USER INTERFACE MULTILEVEL SECURITY ISSUES IN A TRANSACTION-ORIENTED DATA BASE MANAGEMENT SYSTEM

Stanley R. Ames, Jr.

MTP-178



THE MITRE

December 1976

MTP-178

USER INTERFACE MULTILEVEL SECURITY ISSUES IN A TRANSACTION-ORIENTED DATA BASE MANAGEMENT SYSTEM

by

Stanley R. Ames, Jr.

January 1977

THE MITRE

Bedford, Massachusetts

This research was supported by the Defense Advanced Research Projects Agency under
ARPA Order No. 3006
Contract No. F19628-76-C-0001
MITRE Project No. 807B.

ABSTRACT

The requirements for multilevel security have a great impact on the formulation of a user interface, especially in a transaction-oriented data base management system. In this report, we discuss several tradeoffs between the requirements for multilevel security and the requirements for a facile user interface.

TABLE OF CONTENTS

SECTION I

INTRODUCTION

The Department of Defense Advanced Research Projects Agency, the Navy and CINCPAC have agreed to carry out a Military Message Experiment to evaluate computer-aided message-handling systems in an operational military environment. Such systems are merely transaction-oriented data base management systems. Our role in the experiment is to investigate the security ramifications of such message-handling systems. This investigation includes the identification of primitives that are needed to ensure security, and the identification of the impacts that security imposes on the user interface. In this report we discuss several tradeoffs between secure implementation and the richness of the user interface.

BACKGROUND

In response to the need to process multiple levels of classified data, the Air Force Electronic Systems Division sponsored several research and development efforts to build an operating system that would satisfy security requirements in a technically verifiable way. Technical verification was required in order to demonstrate that the system met the security requirements of the Department of Defense.[1] We have borrowed many of the results of the ESD work in the message-handling experiment. Specifically, we are designing our systems to follow the rules of a mathematical model based on the concepts of a reference monitor – an abstract mechanism that controls the flow of information within a computer system by mediating every attempt by a subject (active system element) to access an object (information container).†[2] The hardware-software mechanism that implements the reference monitor is called a security kernel. The security kernel uses the rules of the mathematical model as a specific policy in mediating access requests. This incorporation of policy into the kernel allows for a proof which verifies that the kernel correctly applies the policy to the information it protects.

The mathematical model[3, 4] establishes an "inductive nature" of security by showing that the preservation of security from one state to another guarantees total system security. The model defines security with two rules: the simple security condition and the *-property.[5] The simple security condition states that a subject cannot observe an object unless the security level of the subject is greater than or equal to the security level of the object.†† The *-property further restricts possible

---

† In a computer system, subjects are users and processes, and objects include programs, data files, and peripheral devices.

†† A security level is composed of a classification and a set of compartments. One security level is considered greater than or equal to a second security level if: 1) the classification of the first is greater than or equal to the classification of the second, and 2) the set of compartments of the first is a superset of the set of compartments of the second.

access by stipulating that a subject may not modify an object if that object has a security level lower than the security level of the subject.

At first the *-property may seem overly restrictive. In the manual paper/pencil system we trust all the tools we use not to disclose information. However, we cannot necessarily trust the tools provided by the computer utility, because disclosure of information can potentially happen with great speed, and there exists little chance for catching the violator. The *-property, if rigidly enforced, prevents a service operating on behalf of a user from reducing the sensitivity of any information, placing the responsibility for the classified data on the people who are using it.

To provide security, a kernel must: 1) mediate every access by a subject to an object, 2) be protected from unauthorized modification, and 3) correctly perform its functions. A kernel satisfies requirement (1) by creating an environment within which all non-kernel software is constrained to operate and by maintaining control over this environment. The kernel can be thought of as creating an abstract or virtual machine on a per process basis, i.e. the kernel creates a virtual environment in which a process thinks it is the only user on the machine. This machine performs instructions from the base hardware (i.e., the hardware on which the kernel software runs) and functions implemented by kernel software. If a security policy is correctly built into the abstract machine, programs running on it will not be able to perform operations that violate the policy. In practice, the abstract machine created by a security kernel will include all of the unprivileged machine instructions of the base hardware, constrained by a hardware-supported memory protection mechanism.

The requirement for protection against unauthorized modification is satisfied by isolating the security kernel software in one or more protection domains. As an example, a ring mechanism[6] can be used to provide a domain protected from unauthorized modification. Finally, the requirement that the kernel correctly perform its functions is satisfied by using a formal methodology to demonstrate its correctness. A suitable methodology was introduced by Bell and Burke[7]. Basically, there are two steps: 1) a proof that the kernel behavior enforces the desired security policy, and 2) a proof that the kernel is correctly implemented with respect to the description of its behavior used in the first step. Kernel behavior can be described with a non-procedural program specification. A method for proving that a kernel specification is a valid interpretation of a methematical model of a security policy has been developed by Ames[8], and Millen[9]. Techniques developed by a group at the Stanford Research Institute can be used to prove that the implementation of a kernel (or any other computer program) is correct with respect to its specification[10, 11]. These techniques involve the decomposition of the kernel into hierarchically structured levels of abstraction.

Presently, there are no plans to design and verify a security kernel for the Military Message Experiment. Rather, we are attempting to design the message systems in such a way that we could replace the identified security primitives with a verifiably secure security kernel. We will rigorously scrutinize the designs to ensure that the user interface provided remains unchanged should the message system be built on a security kernel. In addition, the security primitives will be evaluated to ensure that their usefulness warrants inclusion in a future security kernel.

RELATED RESEARCH

The designs of secure data base management systems have been investigated by several other projects, most notably Marvin Schaefer and Thomas Hinke of System Development Corporation and a team from I. P. Sharp Associates Ltd. SDC's work involved the designs of a secure data base management system built on a Multics Security Kernel. Schaefer and Hinke concluded that while all the problems of creation and deletion have not been resolved, a relational data base could comfortably exist within the confines imposed by a multilevel security kernel[12]. I.P. Sharp Associates Ltd. is currently investigating the design of kernel primitives that are related to data base management systems. Neither study has addressed the impact that security has on the user interface. The Military Message Experiment is unique in that the requirements for multilevel security and a usable operator interface are both present.

FACTORS

A primary requirement of the Military Message Experiment is the development of a multilevel secure operator interface. Usability can be judged in three areas: the features provided, the ease in which commands are entered, and the overall performance. Certainly an interface that does not provide needed functions, or is difficult to learn and use, will not be used. In addition, it is clear that a message system that does not exhibit secure behavior cannot be used by the military.

The types of features that we desire to support include the ability to interactively read, create, file, and annotate messages. With approximately a thousand messages being received a day, selective retrieval of a message on the basis of key words, date-time-groups, originators, and subjects is required. Given the necessary features that must be supported by a military message service, a message must be considered a multilevel object. The primary fields of a message are as follows:

- **Header:** The header of a message contains the address, destination, originator, date-time-group identifier, overall message classification, etc. The header is defined to be unclassified.
- **Subject:** The subject of a message may be of any classification less than or equal to the overall message classification.
- **References:** Most references contain a date-time-group which is unclassified; however, certain references may contain additional information which, unless specifically given a classification, must be treated at the overall classification of the message.
- **Text:** The text of the message has a classification usually equal to the overall message classification. Paragraphs of the text may be labeled at a classification that is lower than the overall classification of the text.

- **Annotations:** Although not specifically part of a message, annotations may be added by a user to a message, or message field, at any classification.

- **Key Words:** For filing or retrieval purposes, key words will be added by the user to the message. Key words may be of any classification.

PARTICIPANTS

In an effort to investigate alternative solutions to the problems encountered in message system development, DARPA and the Navy are sponsoring three separate message system development teams representing: Information Sciences Institute, Massachusetts Institute of Technology, and Bolt Beranek and Newman Inc. Throughout this paper, recommended solutions to the various security problems encountered at the user interface are presented. The three development teams have not necessarily selected the solutions given.

SECTION II

SECURITY TRADEOFFS AT THE USER INTERFACE

OBJECT SIZE AND FEATURES SUPPORTED

The size of an object, the basic protection unit that the hardware and software of the system can support, will significantly influence the types of features that can easily be supported. The smaller the size of the object, the larger the number of features that can be supported.

In the Military Message Experiment, we decided that we did not wish to be constrainted by the object size that TENEX, our host system, could support. We based this decision on the fact that TENEX is being used for convenience only. We believe that we can build a kernel that can support whatever object size that is shown to be most suitable to the needs of a transaction-oriented data base management system. By not constraining ourselves to TENEX, the basic protection unit can be determined by the size of object that the terminal can support.

The terminal being used is a modified HP 2645. This terminal can support three different object sizes: the entire screen, a window on the screen, or a domain within a window. Each window consists of one or more lines on the screen and scrolls independently of other windows. Each window is, in effect, a virtual terminal. A domain is a set of one of more contiguous characters within a window; a domain cannot be scrolled independently of other domains in the window.

If we consider the screen to be the fundamental size of the protection object, we achieve the simplest form of the user interface. Each user is able to work at only one security level at a time, which implies the need for only one process per user to be active at any time. Almost no verification of the terminal or the software operating within it is required. With the entire screen at a single security level, the user is informed of his working security level by a single set of security level indicators attached to the terminal.

The primary disadvantage with this approach is that it drastically limits the extent and richness of features that can be supported at the user interface. There are two features in particular that the user desires: the ability to generate a message at a lower security level while reading information at a higher security level, and the ability to have the header information of a message automatically copied on a reply, especially when the reply is done at a lower security level than the text of the message being replied to.[†]

[†] The second disadvantage can be solved for formal message traffic by keeping an unclassified copy of the message header to be used for replies. This solution requires, however, that the message header of newly created messages be reclassified when the message is released.

Choosing the window as the basic protection unit allows increased richness of the user interface. If each window can have an independent security level, generation of messages at one level while reading at a higher level is easy to implement. In addition, having a smaller protection object allows the message systems to treat several parts of a single message at separate security levels. Treating the header of a message as unclassified allows the reply command to copy the header regardless of the security levels of the old and new message. In the Military Message Experiment system, which uses windows as the basic protection unit, the user is made aware of his working security level by a set of security level lights that are activated by the location of the screen cursor.

From a security standpoint, using windows as the basic protection unit increases the complexity of the kernel. For example, the terminal would have to verifiably protect the multiple levels of information within it. In addition, having multiple security levels on the screen implies either that significant sections of the message systems must be verified or that a process is required for every active security level used by the terminal. Since having security levels per window greatly increases the richness of the user interface, we have identified tradeoffs between verification difficulties, performance, and the richness of the user interface.

A window may still be too large an object size. For instance, if key words can occur at each of the different security levels, then each key word requires a separate window. Since windows must occupy at least one line on the screen, and the screen only has 24 lines available, one quickly runs out of lines if the message is broken up into fine-grained objects at different security levels. In addition, since each window scrolls independently, the message system has difficulty in reassigning portions of the screen if each line is a separate window. The solution is to make the domain the basic protection unit. If each domain has a separate security level, windows can be assigned as needed to group domains into logical scrolling groups. In addition, multiple security levels can be used on a single line.

While the advantage of security levels per domain may be great for the user, the effect on the security of the system can be staggering. For instance, instead of using lights driven by the window containing the cursor, the MIlitary Message Experiment system that uses this approach labels the security level of a domain by using unforgeable characters at the beginning of each domain. A terminal-driving program must, therefore, interpret all commands to the terminal to ensure correct operation.

After investigating the three object sizes that our terminal can support, we believe that security levels per window offer the best long-range solution to the problem. Security levels per screen do not offer enough flexibility. Security levels per domain do not appear to offer enough added advantages to warrant the additional portions of the system that needs to be trusted.

COMMAND INPUT

There are several methods for entering commands into the system. These include treating all commands as unclassified, treating all commands at the user's maximum security level, requiring the user to enter commands in windows of different security levels, and having the command automatically transfer the user to the proper security level.

Treating all commands as unclassified has the advantage that the user state[†] can be controlled at the unclassified level. Each process at a higher security level can be made aware of the commands that are entered. There exist, however, commands that have classified arguments; an example of this type is a command that locates all messages that have a specific word in their subjects. We consider this word as a potentially classified argument. We have yet to determine how to handle classified arguments to unclassified commands.

Treating commands at the user's maximum security level eases the determination of the proper classification of the parameters, since the user can examine all of his messages at his maximum level. However, several commands must transmit arguments to the lowest level (unclassified). An example of this type of command is the transmit command. Transmission of messages must occur at the level of the message header, which is unclassified. Unless sizeable portions of the system are verified, or unless user reconformation is required for all commands that have lower-level arguments, the protection policy prohibits the transmission of arguments to the proper level.

Multiple command windows eliminate the problem of determining the proper level of the command by forcing the user to enter the command at the proper level. The use of multiple command windows has two disadvantages: less screen space available to enter and display messages; and increased user interface complexity caused by requiring users to predetermine the proper security level for each command. Because we believe that security considerations should not unnecessarily restrict the user, we cannot recommend this solution.

The most desirable solution is to separate the set of commands into: commands that must be entered at the unclassified security level, commands whose parameters can potentially be at the users maximum security level, and commands that can be entered at whatever security level the user is working at. Command invocation would then automatically establish the security level of the command window so that parameters could be properly entered. Then, unless the user desires to specify that the command operate at a different security level from the predetermined security level, he need not be concerned with selecting the security level of the command or its parameters.

To eliminate the necessity of verifying the entire command processor, we suggest the use of command function keys for those functions that predetermine the security level of the command window. These function keys would transmit directly to the verified code that sets the security level of the comm ' window. The use of function keys eliminates the need to verify user typing and command editing operations. Although a certain amount of additional code must be validated, this solution eliminates most of the security and user interface problems of the other methods of command input.

[†]The user state consists of all information that needs to be preserved over security levels. It includes the current message pointer and the tools being used.

THE CASE FOR A LIMITED WRITE-DOWN CAPABILITY

The strict enforcement of the security model eliminates any possibility of a security compromise, a write-down path through the system that releases information of a higher security level to a lower security level. However, there are several situations in a transaction-oriented data base management system where the user, by following instructions given by the system, can inadvertently compromise small amounts of information.

Consider the following example: A user asks for a list of all his messages so that he can determine all the messages with a subject having word "x" in them. To perform this operation, the user must be at the highest security level of any of the messages he wants examined, which is normally his maximum security level. The enforcement of the *-property forces the results of this examination to be at the level at which the examination was performed, again his maximum security level. Should the user then decide to perform any modification to a message returned by this examination that has a security level lower than his maximum security level, he would have to issue commands at the security level of the message that he desires to modify, and tell the system the unique identification of the message[†]. However, the act of giving the system an identifier told to the user at a higher level is itself a formal *-property violation. It is conceivable that a maliciously written program could use this *-property violation to compromise information. We know of no way, however, to eliminate this type of *-property violation, and we believe that unless the entire system is validated, there is a risk that this type of violation will always be present.

Because *-property violations exist through actions of the user, a case can be made to simplify the user interface in situations where user *-property violations exist. The simplification takes the form of allowing the system, in violation of the *-property rule of the security model, to transmit the unique identifier of the message that the user wants to modify. Although doing this increases the possibility that a security compromise can occur within the system, we believe that we can design sufficient controls, including auditing and restrictions on the amount and type of information passed, to limit the bandwidth of this type of *-property violation so that it is no larger than the violation that otherwise occurs through actions of the user. An important aspect of this limitation is the requirement that only a limited amount of fixed formatted information be transmitted to a lower security level for each user request. Allowing the system to transmit this information can, in some situations, greatly simplify the user interface.

[†] The unique identification is required here because the system is unable to transmit the desired identifier due to the enforcement of the security model.

MULTIPLE PROCESSES AND PERFORMANCE

Each different security level that is being used on the terminal requires a separate process on the host machine. The multitude of processes for security may greatly impact performance. On a traditional system such as TENEX, where process control is handled in software, process swapping takes a considerable amount of time. The solution to the security requirements for multiple processes is utilizing hardware that can efficiently support large numbers of small processes.

ERROR RECOVERY IN AN OPEN LOOP ENVIRONMENT

Several of the implementation designs that we have investigated include the concept of an unclassified process: receiving the majority of the commands, determining the proper security level needed to perform these commands, and then activating a process with that security level so that the commands can be performed. With these designs, much of the control of the message system will be done by an unclassified process. The disadvantage with this approach is that, should an error occur between the unclassified control process and the classified operational process, the classified process cannot ask for clarification[†]. Without clarification, error recovery is difficult. This problem is referred to as the open loop problem.

Presently we believe that the best solution to the open loop problem is to allow the system to close the loop when an error of this type is encountered. Closing the loop improves recovery but impacts security, since a *-property violation exists when the loop is closed. We are working on restricting the amount of information that is required to be transmitted and will employ some form of auditing to prevent over-use.

[†] The *-property rule prevents a process of one security level from requesting information from a process of a lower security level.

SECTION III

SUMMARY

We have reached several conclusions that impact the requirements for a usable secure interface. Among these are the realizations that: object size determines the types of features that can easily be supported, a limited type of write-down may be needed to ensure user acceptance, and, unless process control is handled by hardware, the security requirements for multiple processes may impact performance to an unacceptable degree. Among the problems that require additional effort are: the best way to securely enter commands, and identification of tradeoffs between verifying a portion of code and improved user interfaces.

The first implementation of the message systems will be completed in early 1977. These systems will include secure behavior at the user interface. In addition, each system will provide a detailed design for implementing the system in a verifiably secure fashion. This design will be implemented by late 1977, giving us a conceptually secure system. With these operational systems, we will be in a better position to answer questions regarding the impact security has on the user interface.

REFERENCES

1. "ESD 1974 Computer Security Developments Summary," MCI-75-1, Electronic Systems Division (AFSC), L. G. Hanscom Field, Bedford, Massachusetts, December 1974.

2. J. P. Anderson, "Computer Security Technology Planning Study," ESD-TR-73-51, Volume I and II, James P. Anderson & Co., Fort Washington, Pennsylvania, October 1972.

3. D. E. Bell and L. J. LaPadula, "Secure Computer Systems: Mathematical Foundations and Model," M74-244, The MITRE Corporation, Bedford, Massachusetts, October 1974.

4. K. G. Walter, W. F. Ogden, W. C. Rounds F. T., Bradshaw, S. R. Ames, Jr., and D. G. Shumway, "Primitive Models for Computer Security," ESD-TR-74-117, Case Western Reserve University, Cleveland, Ohio, January 1974.

5. D. E. Bell and L. J. LaPadula, "Secure Computer Systems," ESD-TR-73-278, Volume I—III, AFSC, Hanscom Air Force Base, Bedford, Mass., November 1973 – June 1974.

6. R. M. Graham, "Protection in an Information Processing Utility," *Communications of the ACM*, Volume 11, Number 5, May 1968, pp. 365-369.

7. D. E. Bell and E. L. Burke, "A Software Validation Technique for Certification, Part 1: The Methodology," ESD-TR-75-54, Volume I, AFSC, Hanscom Air Force Base, Bedford, Mass., April 1975.

8. S. R. Ames, "File Attributes and Their Relationship to Computer Security," ESD-TR-74-191, Masters' Thesis, Case Western Reserve University, June 1974 (AD A002159).

9. J. K. Millen, "Security Kernel Validation in Practice," *Communications of the ACM*, Volume 19, Number 5, May 1976, pp. 243-250.

10. L. Robinson and K. N. Levitt, "Proof Techniques for Hierarchically Structured Programs," Computer Science Group, Stanford Research Institute, Menlo Park, California, January 1975.

11. L. Robinson, P. G. Neumann, K. N. Levitt, and A. R. Saxena, "On Attaining Reliable Software for a Secure Operating System," *1975 International Conference on Reliable Software*, Los Angeles, California, April 1975, pp. 267-284.

12. T. H. Kinke and M. Schaefer, "Secure Data Management Systems," RADC-TR-75-266, System Development Corporation, Santa Monica, California, November 1975.